ARTICLE
REPRINT

AR-166

June, 1981

Understand The Newest Processor to Avoid Future Shock

Jack Hemenway, Consulting Editor,
and **Robert Grappel,** Consultant

EDN April 29, 1981

Editorial

iAPX 432: Challenge and opportunity for system designers

Seldom does a new product warrant as much discussion as Intel's iAPX 432. Both before its official February introduction and afterwards, articles in many journals analyzed this new device and its implications. Unfortunately, many of those articles contain errors, perhaps because the product is so complex. Here, we aim to correct such misconceptions and confusion; we'll expand on and reinforce the perspective provided by Jack Hemenway and Bob Grappel in their EDN μC Systems Design Series article, which begins on pg 129.

To begin, we emphasize that the 432 is *not* just a 32-bit μP in the common sense of the term. You can't really call it an ADA machine, either. In fact, even the "micromainframe" designation coined by Intel isn't accurate. Why? Perhaps the answers to 10 questions put the product into better focus:

- **What is the 432? That is, what does it really represent?**

It's a totally new generation of computers, conceived by—and *for*—computer scientists. As such, it represents both a significant educational challenge for EEs and a powerful tool for the system designers who can understand its potentials and use them to the fullest.

- **Can you be more specific about the hardware?**

The three ICs that make up the 432 processor-I/O interface form a pure "black-box" CPU. This CPU's internal structure (in terms of data-path widths, registers and the like) is irrelevant, because there's no way you can gain access to and manipulate any internal elements, as you can in standard μPs. Instead, the physical hardware interface is the familiar Multibus.

- **If the 432's hardware is inaccessible and its interface is an existing standard, what educational challenge does the machine present?**

Writing programs for the 432 is different from writing programs for other CPUs. Instead of considering the machine in terms of bits and registers, you must focus on software objects, which represent a much higher level of

Continued on pg 34

An Award*-Winning Magazine
1978 Staff-Written Series —
System Design Project
1978 Contributed Series —
Designer's Guide to Fiber Optics
1977 Contributed Series —
Software Design Course
1976 Special Issue —
Microprocessor Reference Issue
1975 Staff-Written Series —
Microprocessor Design Series

**Jesse H Neal Editorial Achievement Awards are the business-press Pulitzer Prize equivalent.*

Editorial

abstraction—one familiar to computer-science graduates but perhaps less so to EEs.

In essence, the 432 has no assembly language, per se; you might consider its instruction set to be a high-level language. Actually, though, the instruction set is a nearly optimum intermediate language specifically designed to simplify the task of writing efficient compilers. In turn, these compilers can efficiently handle programs coded in high-level languages.

Note that the 432's "software on silicon" does not constitute a complete operating system; rather, it contains the essential primitives from which you can construct such an operating system. Note also that the machine can't be programmed directly in code written in, say, ADA or FORTRAN; you need the appropriate compiler in each case. Intel currently offers a systems-implementation-language compiler that's a superset of ADA, however, and compilers for other high-level languages will soon become available.

- **What impact does the 432 have on the design process?**

Any good system design begins with the development of a functional specification. For equipment based on most µPs, the translation of this specification into efficient code is a very involved process that includes many levels of complicated detail. By contrast, the 432 eliminates nearly all of this detail, greatly simplifying the programming task.

- **Does the 432 require any special design tools?**

Yes. Intel offers a high-level dynamic debugger. This diagnostic tool is very desirable, but not essential, when working with less sophisticated µPs. However, it's a virtual necessity for executing and debugging programs for the 432. By contrast, some traditional design tools might not prove particularly useful with the 432—for example, attaching a logic analyzer to the 432's data bus might yield information that's very difficult to analyze.

- **What is the 432's basic advantage?**

Because the 432 simplifies system programming, programmers need not be hardware experts; thus, personnel with a lower level of expertise can program it. Further, programs written in high-level languages can generally be developed more rapidly than those coded in assembly language.

- **What are the 432's other key advantages?**

The machine performs arithmetic operations quickly and with a high level of precision. It also automatically prevents many typical programming errors (you can't inadvertently execute data, for example). Further, it provides functional redundancy checking, a feature that allows graceful system degradation when a CPU fails in a multiprocessing environment. And finally, the 432 ensures that all programs are naturally re-entrant and recursive.

- **What disadvantages and limitations does the 432 exhibit?**

The 432 chip set is complex and currently very expensive. It could incur speed penalties when performing certain operations—a matter EDN will investigate in future articles.

- **Is the 432 the ultimate processor?**

Probably not. Although its instruction set is close to the ideal, it might be improved in other ways. For example, although the 432 has an array-type addressing mode, it only gives you one array index. An improved version of the chip set with multiple array indices would allow higher throughput.

- **Will the 432 make popular 4-, 8- and 16-bit µPs obsolete?**

Definitely not, although they could be relegated to attached-processor status in computation-intensive system applications. But remember that such applications could not be handled by µPs at all before the 432's introduction, so the overall market for the older machines should increase. In particular, a significant sales opportunity now exists for Multibus-compatible boards based on processors other than standard Intel devices.

In summary, a good analogy is that the 432 is to standard µPs what 7400 Series TTL was to discrete-device logic gates and flip flops. It should free EEs from many mundane system-design chores, allowing them to concentrate on more rewarding creative pursuits while also reducing their projects' software costs. It's a minicomputer replacement, one that will open up scores of application opportunities.

Although EDN does not endorse the 432 as a product, per se, we do congratulate Intel on its major advance in semiconductor (and computer) technology. And we urge you to consider the part's implications carefully.

Walt Patstone

Walt Patstone
Editor

Understand the newest processor to avoid future shock

You might call Intel's iAPX 432 system a Third Wave machine—it's different from current µPs and mainframes alike. Review its features to grasp its extensive capabilities.

Jack Hemenway, Consulting Editor,
and **Robert Grappel,** Consultant

The vacuum tube, the transistor, the microprocessor—at least once in a generation an electronic device arises to shock and strain designers' understanding. The latest such device is the iAPX 432 micromainframe processor, a processor as different from the current crop of µPs (and indeed, mainframes) as those devices are from the early electromechanical analog computers

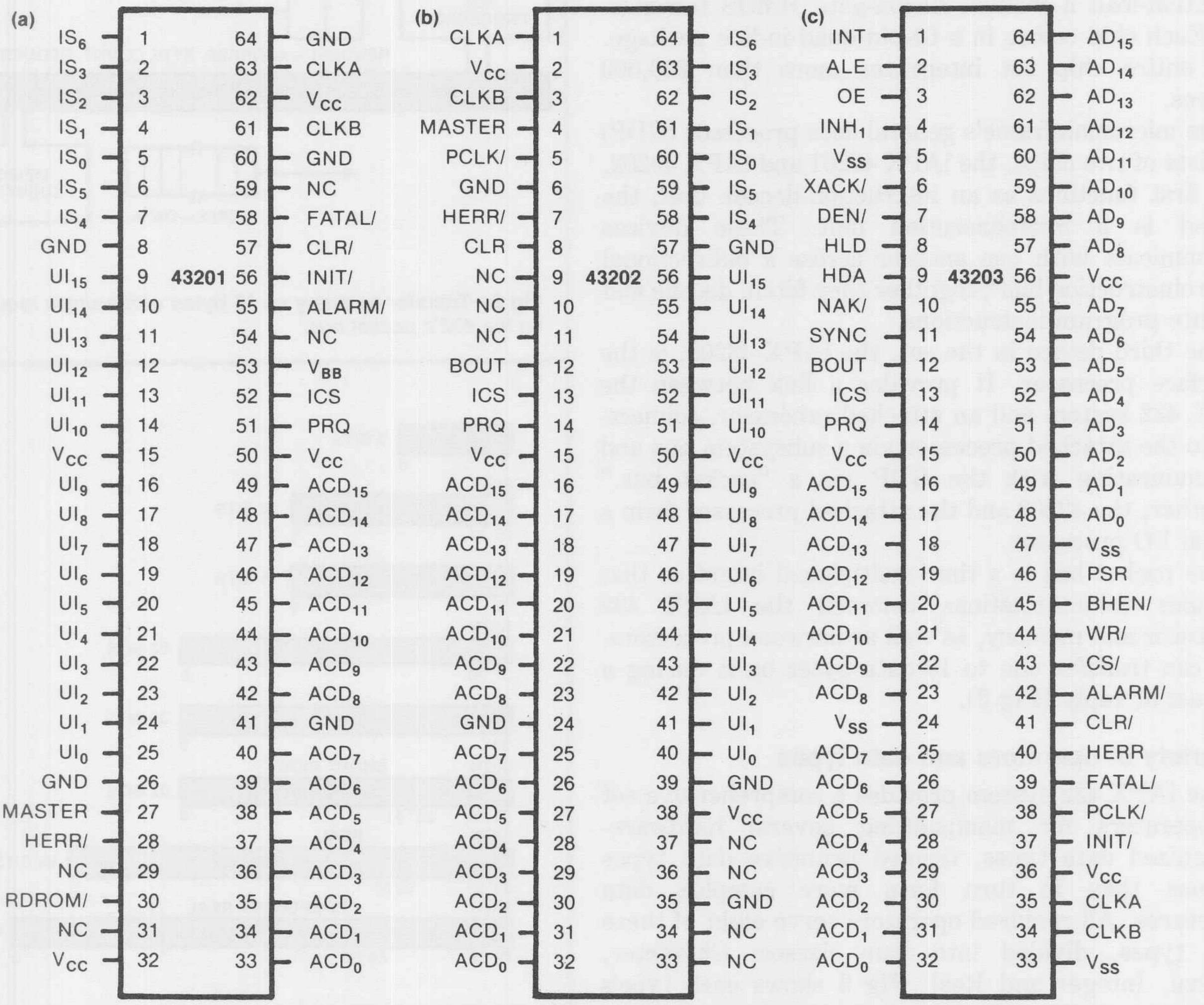

Fig 1—Three chips—*the 43201 instruction-decode unit* **(a)**, *43202 microexecution unit* **(b)** *and 43203 interface processor* **(c)**—*constitute the iAPX 432 micromainframe. Each comes in a 64-pin quad-in-line package.*

of the 1940s. In this article, we describe the highlights of Intel Corp's latest machine; future installments will explore how to use it.

Because the 432 is so different from any processor you might be familiar with, beware of prior assumptions. Some questions you might ask about it could be meaningless, as would asking how many oats the first automobile ate or how much sleep it required. The iAPX 432 product family represents more than a new architecture; it's a proposed system solution to the problem of reducing the cost and development time required for software-intensive μC applications. Thus, in addition to a new architecture, it features a denser VLSI structure than current machines, a new hardware/software mix, a new language (ADA) and much more.

Start with the chips

The basic iAPX 432 processor-I/O interface consists of a 3-chip set **(Fig 1)**, fabricated with Intel's 5V depletion-load n-channel silicon-gate HMOS technology. Each chip comes in a 64-pin quad-in-line package. The entire chip set integrates more than 200,000 devices.

The micromainframe's general data processor (GDP) consists of two chips, the iAPX 43201 and iAPX 43202. The first functions as an instruction-decode unit; the second is a microexecution unit. These devices communicate with one another across a bidirectional "microinstruction bus"; together they fetch, decode and execute program instructions.

The third device in the set, the iAPX 43203, is the interface processor. It provides a link between the iAPX 432 system and an attached processor, connecting to the attached processor via a subsystem bus and communicating with the GDP via a "packet bus." Together, the 43203 and the attached processor form a logical I/O processor.

The packet bus is a time-multiplexed interface that provides communications between the iAPX 432 processor and memory, as well as between processors. You can transfer one to 16 data bytes on it during a request or reply **(Fig 2)**.

A variety of operators and data types

The iAPX 432 system provides a comprehensive set of operators for manipulating several hardware-recognized data types, termed primitive data types because they in turn form more complex data structures. All required operators serve eight of these data types, divided into four classes: Character, Ordinal, Integer and Real. **Fig 3** shows each type's formats.

You can divide the operators for these data types into several broad groups: arithmetic, logical, relational, conversion, move and bit-field manipulation. **Fig 4** shows all hardware-recognized data types and all the operators you can use with them.

The more complex structured data types comprise two forms commonly used ir high-level language (HLL): arrays and records. An array consists of several components, each of the same data type. Thus, there are arrays of integers, characters, and so on. On the other hand, a record consists of several components (usually termed fields) that can represent different data types. Thus, a record might consist of characters, integers and real numbers. (One example might be an employee's work record.)

The iAPX 432 doesn't support these structured types with a set of hardware operations, but it does provide a mechanism that permits their easy manipulation. You can gain access to each of the primitive types through several addressing modes, which facilitate the selection of individual elements from arrays and records. The addressing mode used to reference an operand is determined by the way in which the system forms the logical address in the instruction that references it.

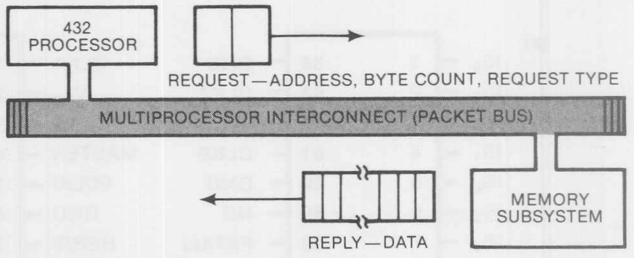
432 PROCESSOR
REQUEST—ADDRESS, BYTE COUNT, REQUEST TYPE
MULTIPROCESSOR INTERCONNECT (PACKET BUS)
MEMORY SUBSYSTEM
REPLY—DATA

Fig 2—Transfer as many as 16 bytes *with a single request on the 432's packet bus.*

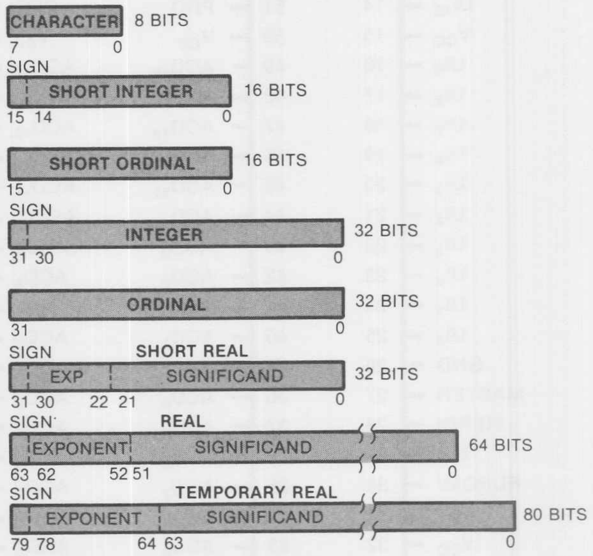

Fig 3—The 432's hardware *recognizes eight primitive data types.*

	OPERATOR	CHARACTER	SHORT ORDINAL	ORDINAL	SHORT INTEGER	INTEGER	SHORT REAL	REAL	TEMPORARY REAL
MOVE OPERATORS	MOVE	X	X	X	X	X	X	X	X
	SAVE	X	X	X	X	X	X	X	X
	ZERO	X	X	X	X	X	X	X	X
	ONE	X	X	X	X	X			
LOGICAL OPERATORS	AND	X	X	X	—	—	—	—	—
	OR	X	X	X	—	—	—	—	—
	XOR & XNOR	X	X	X	—	—	—	—	—
	COMPLEMENT	X	X	X	—	—	—	—	—
ARITHMETIC OPERATORS	ADD	X	X	X	X	X	*	*	X
	SUBTRACT	X	X	X	X	X	*	*	X
	MULTIPLY		X	X	X	X	*	*	X
	DIVIDE		X	X	X	X	*	*	X
	REMAINDER		X	X	X	X			X
	INCREMENT	X	X	X	X	X	—	—	—
	DECREMENT	X	X	X	X	X	—	—	—
	NEGATE	—	—	—	X	X	X	X	X
	ABSOLUTE VALUE	—	—	—			X	X	X
	SQUARE ROOT								X
BIT-FIELD OPERATORS	EXTRACT		X	X	—	—	—	—	—
	INSERT		X	X	—	—	—	—	—
	SIGNIFICANT BIT		X	X	—	—	—	—	—
RELATIONAL OPERATORS	EQUAL	X	X	X	X	X	X	X	X
	NOT EQUAL	X	X	X	X	X			
	EQUAL ZERO	X	X	X	X	X	X	X	X
	NOT EQUAL ZERO	X	X	X	X	X			
	GREATER THAN	X	X	X	X	X	X	X	X
	GREATER THAN OR EQUAL	X	X	X	X	X	X	X	X
	POSITIVE	—	—	—	X	X	X	X	X
	NEGATIVE	—	—	—	X	X	X	X	X
CONVERSION OPERATORS	CONVERT TO CHARACTER	—	X						
	SHORT ORDINAL	X	—	X					
	ORDINAL		X	—		X			X
	SHORT INTEGER				—	X			
	INTEGER			X	X	—			X
	SHORT REAL						—		X
	REAL							—	X
	TEMPORARY REAL		X	X	X	X	X	X	—

NOTES:

X = THIS OPERATOR IS AVAILABLE FOR THE GIVEN DATA TYPE.
* = THIS OPERATOR IS AVAILABLE FOR THE GIVEN DATA TYPE *AND* FOR OPERATIONS WHERE ONE OF THE OPERANDS IS A TEMPORARY REAL.
— = THIS OPERATOR IS NOT AVAILABLE AND WOULD NOT BE USEFUL IF IT WERE.
(BLANK) = THIS OPERATOR IS NOT AVAILABLE.

Fig 4—Operators and data types *for the iAPX 432 provide a variety of manipulation capabilities.*

Instruction set is totally symmetric

Raising the hardware/software interface

The iAPX 432 instructions specify operators and the operands they act on. Some operators require as many as three operands. Instructions reside in special hardware-recognized instruction segments in memory; the GDP views such an instruction segment as a continuous string of bits termed an instruction stream. Instructions can contain a variable number of bits and can begin at any bit within the stream. (In the system's current implementation, the GDP reads an instruction segment in units of 32 bits.)

The 432's general instruction format consists of four fields arranged in the format shown in **Fig 5.** And if you think that this format looks like quads, you're correct. Compiler writers use such quads as an intermediate representation of a compiled program; in traditional machines, the quads then generate the object code. Note, though, that a compiler for the iAPX 432 wouldn't need this last step of code generation; in the 432, the quads *are* the object code.

The instruction format's displacement consists of two subcomponents: a base value and an index value. Each subcomponent can reference its value either directly (the subcomponent contains the value itself) or indirectly (the subcomponent contains a pointer to the value).

A direct reference by both subcomponents is equivalent to a single-component displacement and can serve to gain access to nonstructured data (scalars). Indirect references can combine with such direct references to easily gain access to three structured data types. The four combinations of direct and indirect references are termed addressing modes (**Fig 6**):

- Base and Index Direct—Used to access scalars
- Base Indirect, Index Direct—Used to access records
- Base Direct, Index Indirect—Used to access static arrays
- Base and Index Indirect—Used to access dynamic arrays.

The object-selector logical-address component shown in **Fig 5** can also be specified indirectly—simplifying the creation of large multisegment arrays.

The GDP automatically scales an index value by multiplying it by one, two, four, eight or 16, depending on whether the operand it points to occupies a byte, double byte, word, double word or extended word. The compiler is thus freed from having to perform this calculation.

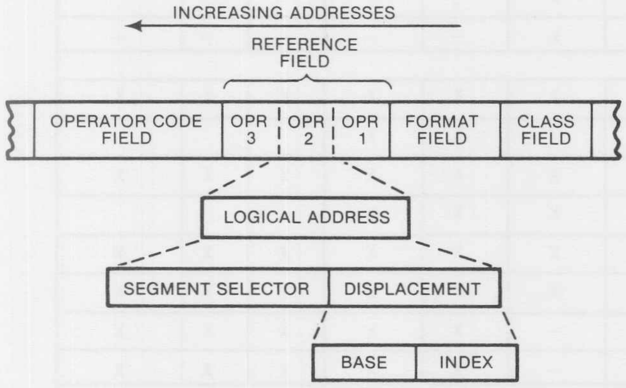

Fig 5—Instruction-format length *in the 432 varies and can accommodate as many as three operands.*

The 432's first application

Intel's iAPX 432 will be used in an iSBC-compatible, board-level evaluation system designated the Intellec 432/100. This system includes an iSBC 432/100 board, which is Multibus compatible and has an RS-232C serial interface, plus object-builder evaluation software and seven introductory texts and references. It is intended as an evaluation tool and functions in conjunction with an Intellec development system. With it, a user can create and execute iAPX 432 code, using a symbolic object-oriented language. The $4250 system is available now. Delivery, 90 days ARO.

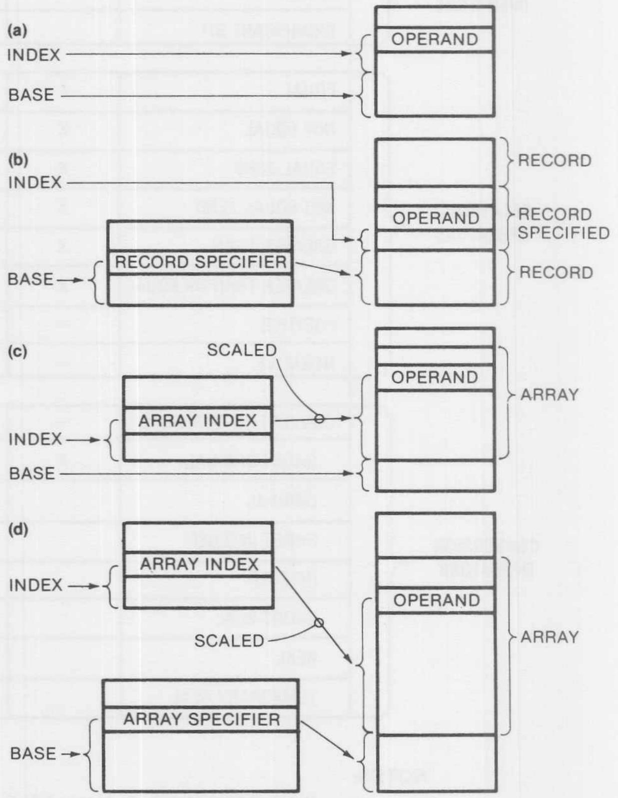

Fig 6—Direct and indirect references *can combine in four ways: Specify the index and base directly to access scalars* ***(a)****, specify the base indirectly to access records* ***(b)****, specify the index indirectly to access static arrays* ***(c)*** *and specify the base and index indirectly to access dynamic arrays* ***(d)****.*

What about floating-point numbers?

Consistent with the IEEE proposed floating-point standard (EDN, July 20, 1980, pg 92), the iAPX 432 allows three floating-point data types: 32-bit short real (single precision), 64-bit real (double precision), and 80-bit temporary real (extended precision). You can implement signed 64-bit arithmetic by interpreting only the fraction and sign bits of temporary-real operands with temporary-real operators—an action possible because the mantissa (significant or fraction) of a temporary-real operand is 64 bits wide.

All floating-point operators perform their operations using 80-bit temporary-real data types. Internally, these values are actually 83 bits long (the extra bits are termed guard, round and sticky bits). All results except those of explicit conversions reside in temporary-real (80-bit) destinations. The Set Context mode operator (instruction) controls precision, rounding and faulting on an inexact result:

- **Precision**—For compatibility with language translators (for example, FORTRAN), the 432 computes results to a programmer-controlled precision (24, 53 or 64 significant bits).
- **Rounding**—If the 432 can't exactly represent a result in the specified precision, it rounds the result to one of four modes specified by the programmer (Up, Down, Toward Zero or To Nearest).
- **Faulting on inexact results**—The programmer can also specify that a fault should occur on an inexact result—one that can't be exactly represented in the specified precision.

Additionally, the 432's hardware detects several reserved values termed NaNs (Not a Number); each value produces an Invalid Operand fault. Similar in concept to trapping, this faulting process invokes user-supplied software to handle the reserved value (positive infinity, for example).

To manipulate floating-point operands, the 432 provides such operators as data-transfer, arithmetic, relational and conversion instructions. You even get square-roots and absolute values.

The iAPX 432 instruction set is completely symmetric: All required operators are available for every data type, and all four addressing modes are available for any instruction operand. Such symmetry proves especially important for easy translation of high-level languages; without it, a compiler writer faces many special cases in which software must make up for the holes in the instruction set. In such a situation, the compiler is more difficult to write and complicated.

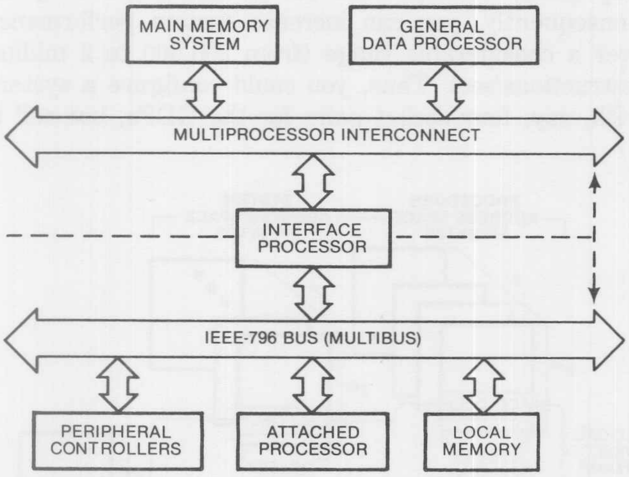

Fig 7—With the aid of the Multibus, *a satellite subsystem (attached processor) can independently handle all I/O.*

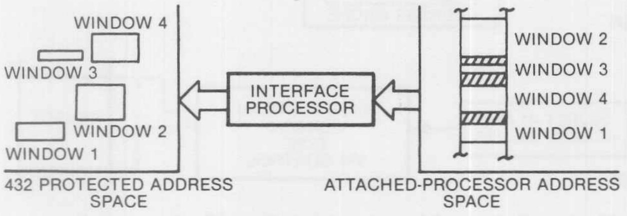

Fig 8—An attached processor *has access to main memory through windows that protect portions of memory not needed for the processor's operation.*

As examples of the efficient coding of high-level-language statements possible with the 432, consider the following statements, each of which is encoded by one iAPX 432 instruction:

A=B*C	A 3-address multiplication, all scalars
A[I]=B[J]/C[K]	A 3-address division, all static-array elements
P.Q=A[I]*C	Three address, mixed type, "element Q of record P is assigned the product of static array A, element I and scalar C."

The 432 also provides stack-related advantages. For example, when an instruction references another instruction, it can explicitly specify a logical address or can gain access to the top of the operand stack—a special data segment maintained by the hardware for expression evaluation.

The act of gaining access to an operand on the stack is termed an implicit reference; it adds or removes items from the "top" of the stack on a LIFO basis. (A hardware-maintained stack pointer points to the current stack top.) Using such implicit stack references rather than explicit memory references to address operands allows shorter and faster instructions. And judicious use of both the 432's stack arithmetic and memory-to-memory arithmetic gives programmers the best of two worlds.

Real-time operation won't slow the system down

I/O subsystem uses the Multibus

Turning now to the 432's I/O communication, we note that the 43203 interface processor (IP) permits a satellite subsystem to act as an attached processor (AP) that independently handles all I/O activity (**Fig 7**). Such an AP is an independent microprocessor, configured from Intel's line of midrange µPs, peripheral controllers and Multibus-compatible devices.

The IP provides protected address "windows" between the AP and the iAPX 432 system memories (**Fig 8**). It also provides a DMA-like buffering function to lower the micromainframe's memory-access overhead. By adding more such AP subsystems or by reconfiguring an existing one, you can greatly expand the iAPX 432's I/O capacity. That is, one system can grow from a few CRT terminals, printers and mass-storage subsystems to a much larger system in a fully compatible manner.

For example, consider a manufacturer of intelligent terminals used in a cluster in which one terminal acts as the master. If this firm acquires a new application, one that requires intensive computational capabilities and perhaps concurrency, what can it do if it can't meet the new requirements with current hardware? The manufacturer can configure a network of terminals with an iAPX 432 at the hub providing the intensive computational and concurrency facilities. The beauty of this approach is that almost all the software in the terminals remains usable.

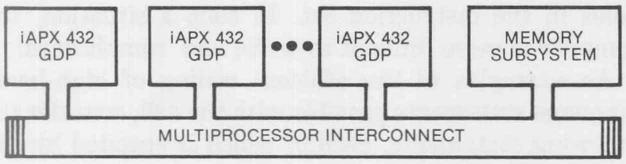

Fig 9—Transparent multiprocessing *requires no software changes as you add data processors to a 432-based system.*

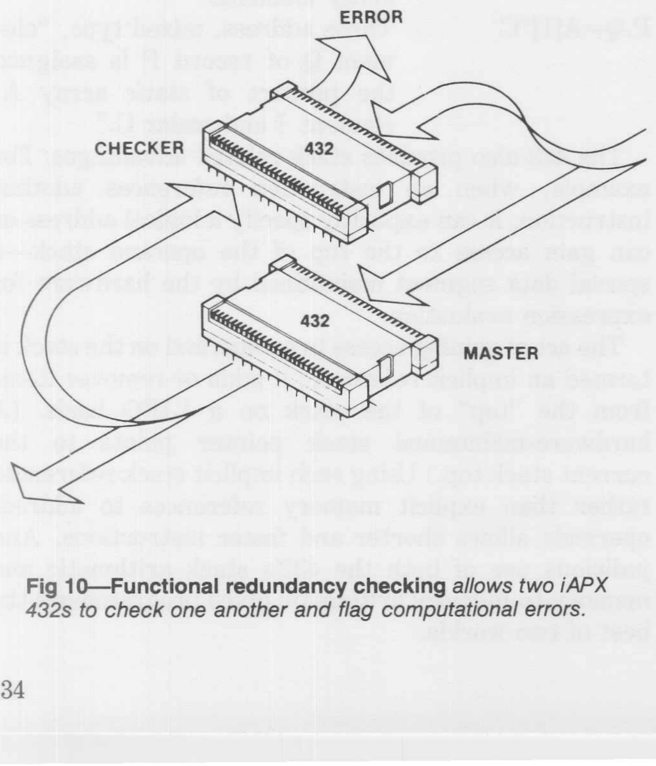

Fig 10—Functional redundancy checking *allows two iAPX 432s to check one another and flag computational errors.*

What about real-time operation?

At this point, you should be wondering why the 432 GDP has no interrupt facilities. The answer is important: The iAPX 432 system is designed so that real-time constraints can't proliferate throughout the system.

After all, it really makes no sense to allow the computation-intensive GDP to be interrupted by real-time signal levels. Instead, keyboard interactions, for example, are handled by an AP that gets interrupted by the keyboard whenever a user depresses a key. The AP then reads the keyboard and stores the corresponding character in a buffer. When the user enters a Carriage Return, it, too, is stored. But before executing a Return from Interrupt instruction, the AP sends a message to the GDP informing the GDP that a message is waiting for it. The GDP can then process the line of text.

Multiprocessing is transparent

The 432 also exhibits other advantages in the area of system expansion. For example, you can routinely expand the central processing system, made up of GDP chip pairs, without software modifications (**Fig 9**). Consequently, you can increase system performance over a considerable range (from 200,000 to 2 million instructions/sec). Thus, you could configure a system with, say, four socket pairs for the GDPs, but sell it

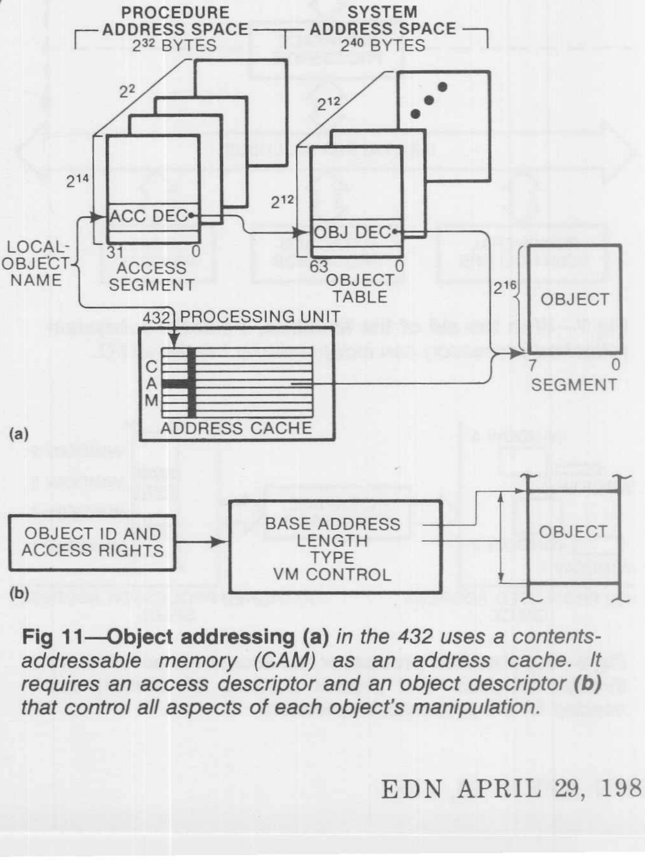

Fig 11—Object addressing (a) *in the 432 uses a contents-addressable memory (CAM) as an address cache. It requires an access descriptor and an object descriptor* **(b)** *that control all aspects of each object's manipulation.*

initially with only one GDP. Then, to obtain more processing power, a customer could merely add GDPs.

Additionally, you can check hardware failures in a 432-based system by wiring any two identical iAPX 432 processors together into a self-checking configuration (**Fig 10**). The devices run at full speed and check for on-chip defects as well as connection flaws. If they don't agree on a computational result, they stop and alert the rest of the system that they can no longer guarantee correct operation.

For example, you could combine this facility with transparent multiprocessing to provide graceful degradation for systems that must stay on the air at all costs. Specifically, you could start with four GDPs and four other checking GDPs. Then, if a fault appeared in one of them, you'd disconnect it from the system. Of course performance would be impaired, but the system would stay up.

A gigantic structured memory

The iAPX 432 has a segmented memory. However, the differences between this processor and other segmented machines are great enough to justify the use of a new term, "structured memory," to describe the 432's memory architecture.

First, the 432 can address a very large number of segments (2^{24}, or approximately 16 million). Each segment can be up to 2^{16} bytes long. The total virtual-address space is therefore 2^{40}, or more than 1 trillion, bytes.

Second, the 432 uses a 2-step mapping process that separates segment relocation from access control. This 2-stage mapping also divides protection features into segment-specific protection (segment type checking) and user-specific protection (reference rights). The process is unique to the 432 and is responsible for some of its most powerful features.

How would you use the 432's gigantic memory? The machine is designed to accommodate virtual-memory techniques. Virtual memory (VM) is a way of implementing a large logical address space within a smaller physical space (EDN, April 15, pg 60). Because no iAPX 432 user will actually attach 1 trillion bytes of main memory to a system, the 432's VM mechanism thus permits use of the address space in a virtual manner.

IBM has implied that a paging mechanism is the only way to implement VM, but there are other ways, and the iAPX 432 uses one of them: a segmented approach. Each segment has an object descriptor containing (among other things) four 1-bit fields. The hardware maintains these fields, which the operating system can use to implement VM:

- The Valid bit—Indicates whether the segment is currently in main memory
- The Storage Allocated bit—Indicates whether any memory has been associated with the descriptor
- The Accessed bit—Indicates whether the segment has been accessed by some executing process
- The Altered bit—Indicates whether the information contained in the segment has been modified by some executing process.

The operating system can use the Valid bit and the Storage Allocated bit to detect when a physical segment is not present in memory, and it can use the Accessed and Altered bits to decide which of the currently present segments should be swapped out or merely overwritten by the new segment. In addition, the OS can use several fields in the segment descriptor to record other useful information regarding the segment (eg, frequency of use) that can also be used in the swapping algorithm.

An object-oriented architecture

The micromainframe's hardware, operating system and system programmng language all support the concept of "objects"—variable-length data structures that have a higher order than data typically recognized and manipulated by contemporary computer hardware (**Fig 11a**).

System programmers write program modules based on objects. Each object is typed by a special "object

EXAMPLE MECHANISMS	EXAMPLE POLICIES
PROCESSORS AUTOMATICALLY DISPATCH PROCESSES BASED ON PRIORITY } PARAMETERS DEADLINE TIME SLICE	REAL TIME, TIME-SHARING, BATCH AND/OR DYNAMIC LOAD SHARING CAN BE OPTIMIZED BY SOFTWARE POLICIES SETTING PARAMETERS
MEMORY ALLOCATION TOTAL STORAGE } PARAMETERS MAX INCREMENTS	MEMORY ALLOCATION PROCEEDS AUTOMATICALLY UNTIL BOUNDS EXCEEDED. SOFTWARE POLICIES THEN HANDLE REQUEST FOR MORE
PROTECTION VIA "NEED TO KNOW" RIGHTS; EG, READ, WRITE, MODIFY, AMPLIFY COPY, DELETE, NO ACCESS	SOFTWARE POLICIES CAN BUILD RANGE OF PROTECTED SYSTEMS FROM FULLY OPEN SHARED SYSTEMS TO SECURE, NEED-TO-KNOW PROTECTED SYSTEMS

Fig 12—Careful separation of operating-system mechanisms *(in silicon) and policies (in software) allow you to optimize applications.*

intel®

INTEL CORPORATION, 3585 SW 198th Avenue, Aloha, Oregon 97007 • (503) 681-8080

Printed in U.S.A./Y55/25K/0681/AP

Virtual memory can span 1 trillion bytes

descriptor" and can only be physically addressed by a special "access descriptor" **(Fig 11b).** What may gain access to the object and what operations may be done to it are inherently controlled. This scheme is the key to the 432's granular data- and program-protection features.

Such an object-oriented architecture supports clean modular programming. You can break down software systems into modules that permit true isolation of design decisions. These modules are separated from one another by interfaces that define what each module does, hide the module's design details from other modules and define the intermodule relationships to ensure an anomaly-free adaptation.

The operating-system kernel is in silicon

The iAPX 432's multiprocessing capability is a product of its silicon-based operating system. That is, operating-system functions such as process scheduling and dispatching, interprocess communication, processor management, storage allocation, memory management, protected-data sharing, exception detection and the like are part of the iAPX 432 hardware, but the policies that govern these mechanisms are separated and applied via system software **(Fig 12).** As a result, system performance increases with no loss in implementation flexibility.

Additionally, the iAPX 432 hardware can recognize system data structures. The operating-system features thus permit processors literally to find their own work, thereby sharing the workload automatically and dynamically.

ADA, a programming language developed through cooperation among the US Dept of Defense, industry and universities (EDN, January 5, pg 171), is the micromainframe's system programming language **(Fig 13).** It's a language oriented toward systems programming, numerical problem solving and real-time applications involving concurrent execution requirements. ADA combines PASCAL's simplicity and elegance with the structure and expressive capability necessary for multifunction software systems. It was selected as the iAPX 432's system programming language because it directly supports object-oriented programming and is expected to further improve productivity for software-development teams.

In fact, ADA *is* the iAPX 432's assembler; the translation from ADA statement to machine instruction is essentially a one-to-one process. As a result, there's no intervening software level to get in the way, as there is in compilers for other machines. EDN

Is the iAPX 432 a 32-bit machine?

Like contemporary mainframes, the micromainframe handles 32-bit data words. But it also operates on floating-point values 32, 64 and even 80 bits wide. It multiplies 32-bit integers in 6.25 μsec and 80-bit floating-point numbers in 21.125 μsec.

True, the data path in the iAPX 432 is only 16 bits wide. But current wisdom has it that the IBM 360 Series computers are 32-bit machines, even though the low-end models fetch only a byte at a time and the midrange units 16 bits at a time—only the high-end models fetch 32 bits at once. Programmers are not generally aware of this feature and so treat the computer as a 32-bit machine no matter what its model number.

Similarly, Intel maintains that if you are willing to consider the IBM 360s as 32-bit machines, the iAPX 432 is also a 32-bit machine.

- **MATCHES DESIGN METHODOLOGY**
  - — BASED ON THE CONCEPT OF OBJECTS
  - — SIGNIFICANT SUPPORT FOR MODULARIZATION
  - — AIMED AT REDUCING PROGRAMMING COSTS
- **CONSTRUCTS MAP THE ARCHITECTURE AND OS**
  - OBJECT ↔ OBJECT
  - PACKAGE ↔ DOMAIN
  - ACCESS ↔ ACCESS DESCRIPTOR
  - SUBPROGRAM ACTIVATION ↔ CONTEXT
- **FEATURES PROVIDE DIRECT ACCESS TO HARDWARE**
  - — 432-SPECIFIC OPERATIONS ARE IN ADA'S STANDARD MACHINE ACCESS PACKAGE
  - — SIMPLE 432 EXTENSIONS TO ADA SUPPORT DYNAMIC SYSTEMS

Fig 13—The advantages of ADA *as the 432's systems programming language are several.*

References

1. Intel Corp, "Introduction to the iAPX Architecture," 171821-001.
2. Intel Corp, "iAPX 432 GDP Architecture Reference Manual," 171860-001.
3. Intel Corp, "iAPX 432 Object Primer," 171858-001.
4. Rattner, J, and Cox, G, "Object-Based Computer Architecture," *Computer Architecture News,* October 1980.
5. Wegner, P, *Programming With ADA: An Introduction by Means of Graduated Examples,* Prentice-Hall Inc, Englewood Cliffs, NJ, 1980.

Author's biography

Robert Grappel is vice president of Hemenway Associates Inc, Boston, MA.